



Lockheed Martin
Advanced Concepts Lab

Bots-R-Us: A Realtime Solution for Defending Super Mario Bros. Against Non-Human Players

Adam Riffel

Senior

Iowa State University
ajriffel@iastate.edu

Carlos Acuna

Senior

Iowa State University
cacuna@iastate.edu

Corbin Graham

Senior

Iowa State University
cgraham1@iastate.edu

Jose Medina Mani

Senior

Iowa State University
jomedman@iastate.edu

Nhan Tran

Senior

Iowa State University
nhtran@iastate.edu

May 2024

Abstract: This report describes our team's findings when exploring the feasibility of automated bot detection in Super Mario Bros. from a naive approach. In it, we will describe the challenges associated with human-bot detection and identification, current solutions, and approaches. We will then discuss our findings with respect to data collection, data processing, and prediction. We will also discuss the importance and impact that this solution could have on multiple domains. Finally, we will explore the feasibility of scaling our solution to multiple domains and multiple users and provide suggestions for continued research, implementation, and product deployment.

Contents

1 Introduction3

2 Background3

3 Objectives4

4 Methodology4

5 Findings5

6 Proposed Solution7

7 Implementation Plan8

8 Implications10

Conclusion10

References11

1 Introduction

Specific forms of AI can be difficult to distinguish from human intelligence. The 2012 paper on Turing track Mario AI provides examples of AI specifically developed to model human behavior, presenting a new problem in detecting such AI ^[14]. The game used for this competition is Super Mario Bros, a simple 2D platformer in which the player can move left or right, jump, run, and collect powerups to pass a variety of obstacles and reach the end of a level within a short timeframe. In this competition, various AI were created by participants to closely resemble the playstyle of human players.

This raised the question of, if it's possible to create an AI to mimic human behavior, can we create something that specializes in differentiating these AI from real people. This question applies not only to cheating in online games but also to the possible future security of machines being accessed by AI attempting to stay unnoticed by mimicking typical human behavior. This is why we researched and developed a framework for machine learning models trained to detect these AI mimicking humans.

2 Background

Previous approaches to bot detection and classification typically focus on two aspects: server-side detection and some type of performance evaluation. For example, in the paper "Bot Detection in Online Games" by Phou Lee, their approach consisted of a server-side focused solution that would watch for the performance of a player ^{[3][9]}. Our approach is unique in that we focus on client-side detection and IEE Turing-tested AI attempting human-like behavior in addition to performance. The reason is that in games like Super Mario Bros, there are players who compete to reach the fastest times in beating the game possible. These players may have little variance in their playstyle with exceptional performance without being a bot, thus making decisions based solely on performance could incorrectly label skilled humans as a bot.

Other approaches are specific to a game or type of game as well where players are grouped into categories by play style, or activities that not all games possess such as social interaction ^{[1][2]}. We plan to make a general solution by using little information on what the player is doing in the game, but how they play based on their input into the game such as buttons pressed.

3 Objectives

Our primary goal was to answer the given research question of *“Is it possible to differentiate between a human and bot playing a game?”* Alongside this primary goal, we also answered the following secondary questions: 1. Can we do this from a naive approach? 2. Can we identify a human or bot in real time? 3. Can we expand this to identify individual users? 4. Can our solution be generalized to multiple platforms? To solve these problems, we evaluated data from the game.

4 Methodology

To distinguish between a bot and a human, we used a variety of machine learning techniques applicable for classification. These ranged from the simple models like logistic regression, support vector machines, and random forest classifiers to once state of the art techniques like Long Short-Term Memory (LSTM) RNN and Hidden Markov models (HMM). Our best performing models were the LSTM Autoencoder and the HMM, but the LSTM came at the cost of requiring large amounts of samples.

To collect both training and testing data, we devised a pipeline to dump the memory of the program, in this case Super Mario Bros., to CSV files that were used to train our models. The dump also contained a label so that we could perform supervised learning tasks using the gathered data.

Once the data was collected it was analyzed to see if any features would provide us with better performing models. Previous research showed that for a modified version of Super Mario Bros. certain features were correlated with player skill ^[14]. We also utilized this method of plotting the average and standard deviation values for certain features gathered through the memory dump. One idea that can be further explored here is the idea of player improvement trajectories, which will be explained in our findings.

5 Findings

First, after performing the feature analysis on our gathered samples we found that certain features did in fact improve the performance of our prediction models.

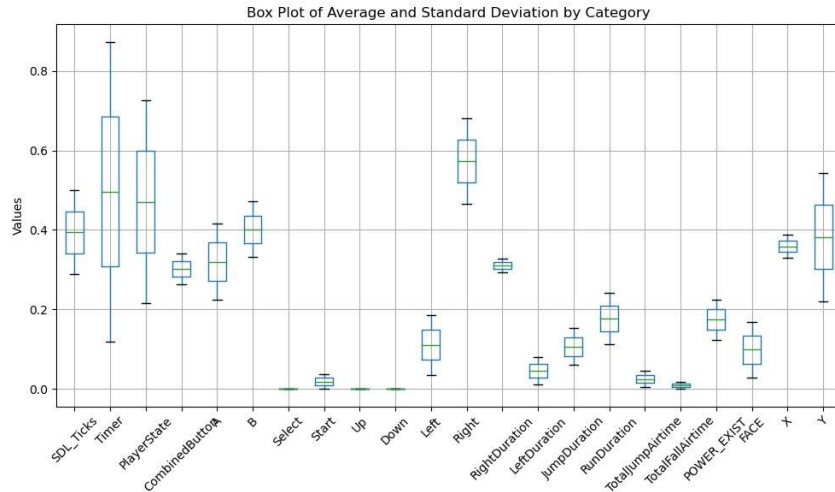


Figure 1: Normalized value ranges for each feature.

Using these graphs, we could also visually identify an improvement trajectory for a particular player. Something to explore further would be using the difference between these values to form an improvement vector and use that to train and test models. The downside would be that it requires much more sample collection than our implemented solution.

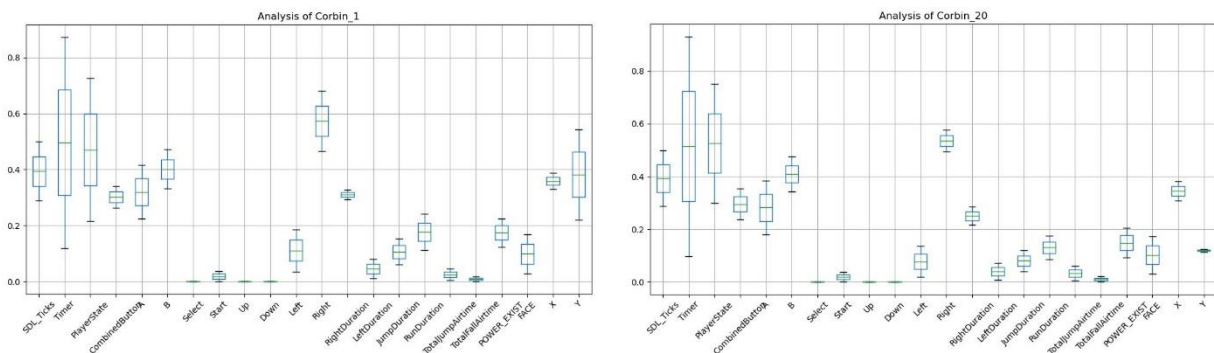


Figure 2: Value ranges of first and 20th run of human playing Super Mario Bros.

One main trend seen across our findings was that utilizing some form of time warping improved the performance of the models drastically. Time warping, as we use it, is the simplifying of our sequences into shorter sequences, with respect to time. This does come with the risk of losing information. One attempt was to use averages and totals to simply a sequence into a single point. Which allowed a K-means clustering model to be run on our data.

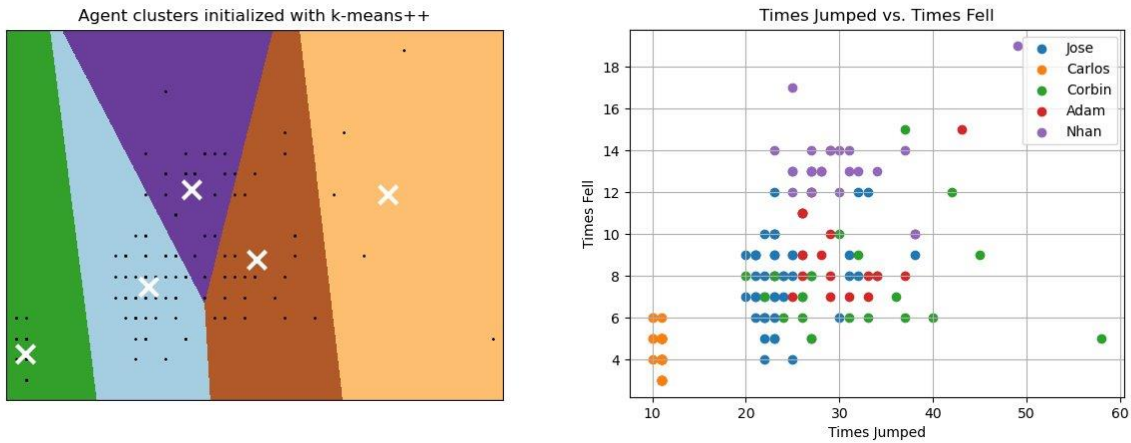


Figure 3: K-means clustering on total number of times jumped.

LSTM Autoencoder encodes the input sequence and tries to reconstruct the sequence. Loss between the input and reconstructed sequence is Mean Average Error (MAE). In the result graph, the red area is the bot gameplay reconstruction error, and the green area is the human gameplay reconstruction error.

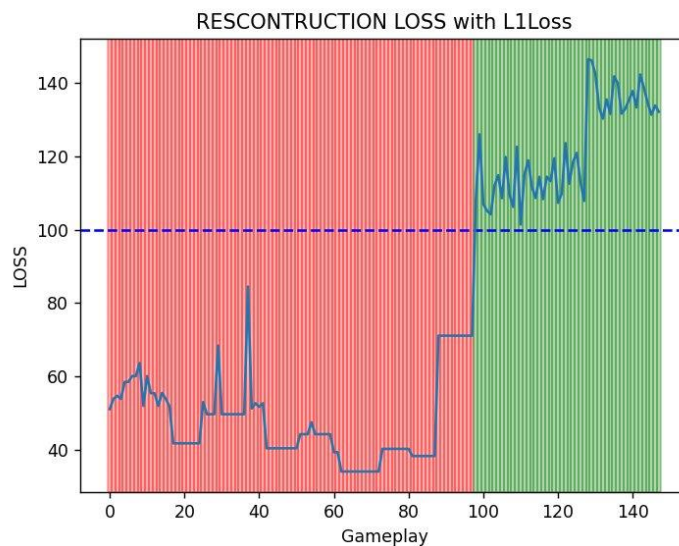


Figure 4: LSMT Autoencoder confusion matrix.

All these models requiring unique forms of data processing to give their best results lead us to our most important finding. There is no feasible way to generalize one model for multiple games or to expand one model for individual identification. This leads us to what we believe is the most practical way of addressing the bot identification problem for multiple domains.

6 Proposed Solution

Our findings have proven it possible to practically differentiate between human and non-human players in the platform game, Super Mario Bros. The difficulty now is to find a way to practically apply our findings in an online scenario.

We propose a framework that manages each of our solutions in a way that allows our client to monitor and adapt the framework for multiple domains and multiple players. This framework would host our three primary components in a nondependent style that allows modification and adaptation as the online environment changes. The three primary components will be demonstrated by three modules: Memory Scraping Module, Data Processing Module, and Prediction Algorithm Module. A complete design of this framework with implementation is shown below in Figure X.

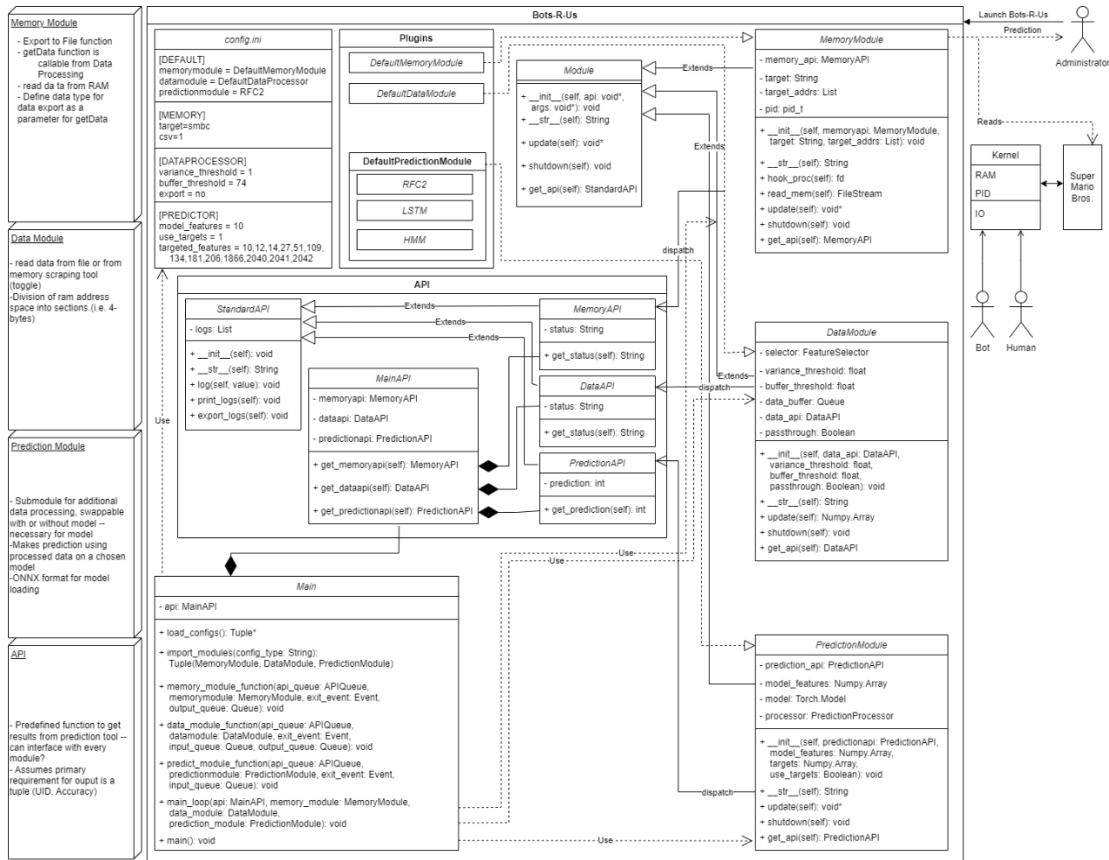


Figure 5. Class diagram of proposed framework implementation.

The proposed framework uses multiple processes to host each of the modular components and multiprocessing queues to transfer information and outcomes. There is a global API that stores individual APIs for each of the modules and is continuously updated as information is being processed. The client can access this global API, Main API, and monitor the status, or protocols implemented in each module.

The proposed framework when implemented demonstrated the ability to manage these modules and provide real-time, online prediction outcomes. It can also easily switch between different memory scraping modules for different applications and different prediction algorithm modules for different prediction approaches.

The potential vulnerabilities of this approach are 1. Specific domain knowledge requirements; the memory scraping module requires some level of domain knowledge about how and where the memory is being stored for a given application. 2. The data processing module allows the client flexibility with data processing but there is a set requirement for data formatting to constrict mismanagement of resources. 3. The prediction module can be trained online but requires immense resources and it is recommended a pre-trained model be used. 4. The framework supports only a single user per instance and would require immense resources to support more than one user. It is recommended the framework be run in a case-by-case scenario where it is only using the resources of the host (or target player). 5. To offer maximum flexibility, and extensibility, the framework is not space efficient. A space efficient design would require far tighter restrictions on modules that would hurt the ability of the framework to adapt to multiple domains and users.

Potential Application. One potential application of the proposed framework is to monitor a user's activity on their device. Once the prediction algorithm has been trained to understand the user's typical activity and threshold, any activity outside of their regular use could be a potential intrusion. What separates our proposed framework from other resources is 1. It does not require any additional hardware. 2. It can continuously monitor the user and their activity, providing online, real-time prediction updates. 3. The framework can monitor an entire domain and is not application limited; it does not require a great understanding of how a system works to begin generating predictions. By leveraging these, we can warn the user, or an administrator if their system is being manipulated without direct monitoring.

7 Implementation Plan

The proposed framework was implemented using Python 3.11. The choice of language was to offer the greatest flexibility and extensibility between multiple models, domains, and users. It is currently in an Alpha version, where it has limited features, insight, and performance. Development was performed using Agile in a continuous integration and deployment environment.

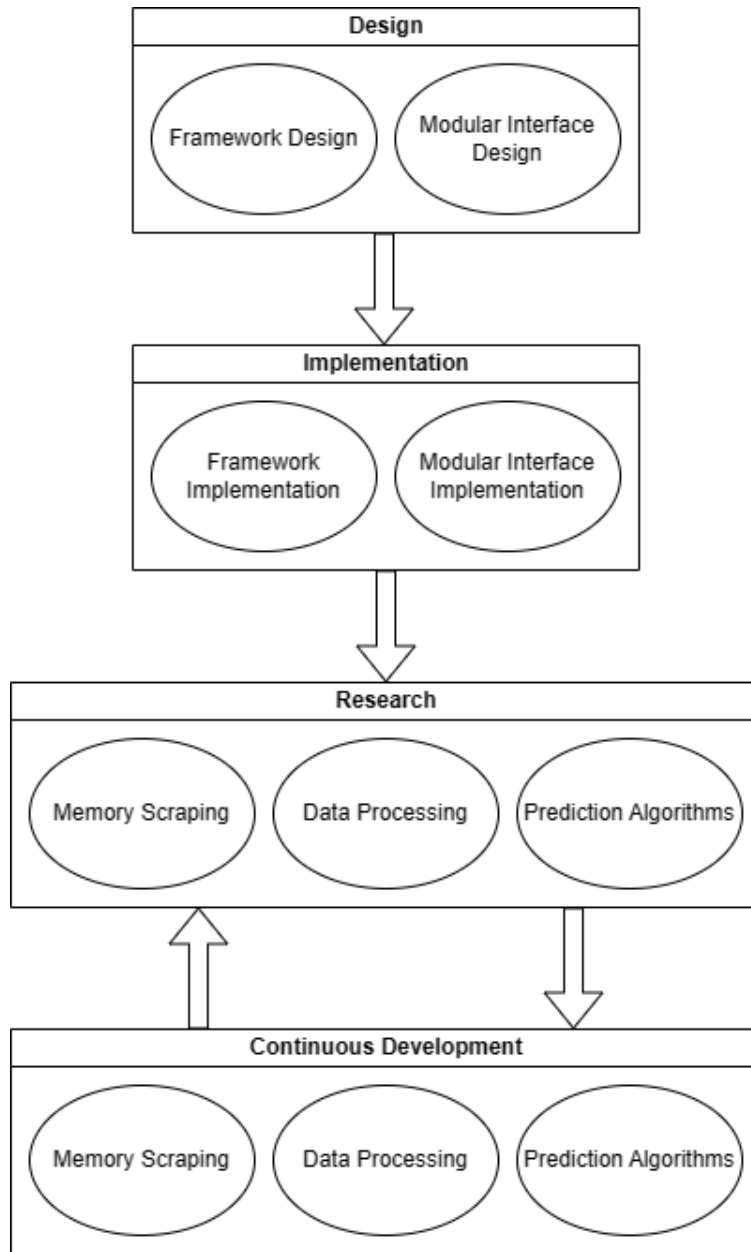


Figure 6. Example implementation roadmap.

To move the framework from conceptual implementation to real world application would require extensive development. The proposed methodology for implementation is to continue the use of Agile with these variations: 1. Teams are split by module where a given team of developers is only aware of their required IO, but not about the functionality or capabilities of other modules. 2. Deployment will happen in modular settings where once the framework design is finalized, only the modules that implement the three primary features must be updated. The framework can continue running without downtime to swap or upgrade a module.

Cost. The cost of the proposed implementation is described below. It assumes hardware, software, and tools are a non-added cost.

Resource	Cost
Research Labor	\$0 Initial (Completed) + \$XX.XX Annual
Development Labor	\$XX.XX Initial + \$XX.XX Annual

8 Implications

The proposed implementation plan and solution would allow for a continuously evolving framework that can provide online anomaly detection and help an administrator determine if a user's system has been compromised. Video game developers can use our solution on the client side to detect and deter the use of bots before data has even begun streaming to servers in an online multiplayer game. It would allow for faster action and response from the administrators and require less administrative input when preventing and negating cheating.

Conclusion

Our study has confirmed that it is not only possible to differentiate between bots and humans in a platform game, Super Mario Bros., but that it can be performed online in real-time. We have proposed the design and development of a framework with a modular design that supports the continuous research, development, and implementation of our findings and supports future research and improvements in methodology.

References

- [1] A Behavior Analysis-Based Game Bot Detection Approach Considering Various Play Styles, <https://arxiv.org/pdf/1509.02458.pdf>
- [2] Multimodal game bot detection using user behavioral characteristics, <https://springerplus.springeropen.com/articles/10.1186/s40064-016-2122-8#:~:text=Server%2Dside%20detection%20methods%20are,method%20for%20detecting%20game%20bots>.
- [3] Bot Detection in Online Games, <https://umm-csci.github.io/senior-seminar/seminars/fall2013/Lee.pdf>
- [4] How to Compare Machine Learning Models and Algorithms, <https://neptune.ai/blog/how-to-compare-machine-learning-models-and-algorithms>
- [5] CIFAKE: Image Classification and Explainable Identification of AI-Generated Synthetic Images, <https://doi.org/10.1109/ACCESS.2024.3356122>
- [6] FakeSpotter: A Simple yet Robust Baseline for Spotting AI-Synthesized Fake Faces, <https://doi.org/10.48550/arXiv.1909.06122>
- [7] Deep learning and multivariate time series for cheat detection in video games, <https://link.springer.com/article/10.1007/s10994-021-06055-x>
- [8] Mario Reinforcement Learning Implementation, <https://www.kaggle.com/code/ratthachaneut/aic502-mario-rl>
- [9] Classification of Humans and Bots in Two Typical Two-player Computer Games, <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8463277>
- [10] Turing Test Framework for Cooperative Games, <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9893684>, <https://github.com/bic4907/Multiplayer-TuringTest>
- [11] Super Mario Bros C, <https://github.com/MitchellSternke/SuperMarioBros-C>
- [12] Intelligent and Adaptive Web Data Extraction System Using Convolutional and Long Short-Term Memory Deep Learning Networks, <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9523501>
- [13] User Identification through Hidden Markov Model-based Touch Keystroke Dynamics, <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10162120>
- [14] The Turing Test Track of the 2012 Mario AI Championship: Entries and Evaluation, <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6633634>

[15] Mario AI Competition,

<http://julian.togelius.com/mariocompetition2009/gettingstarted.php>

[16] Super Mario Evolution, Genetic AI, <http://julian.togelius.com/Togelius2009Super.pdf>

[17] A Hierarchical Approach to Generating Maps Using Markov Chains,

<https://ojs.aaai.org/index.php/AIIDE/article/view/12708/12556>

[18] User Identification through Hidden Markov Model-based Touch Keystroke

Dynamics, <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10162120>